Generating Artificial Imbalance to analyze the impact on Model Performance using Classifier Tuning, Data Reduction, and Time Optimization techniques (MLDM 2022)

Josh Trivedi [*1]

Abstract

This project has been carried out using the waveforms dataset from UCI archives and has 5000 data points, 3 Classes of waves and 21 attributes inclusive of noise. A fourfold approach is carried out which includes tuning a KNN Classifier, Reducing the Complexity and Speeding up the calculation.

1. Introduction

The project files are available on the github repository names $Machine_Learning_Project$ at:

`https://github.com/joshtrivedi/`

Once the code is run, observations are made, on which we base our whole experiment. Machine Learning is the field to study and create algorithms with the ability to learn from observed data, without being explicitly programmed. The k-Nearest-Neighbors is a method for classification that we have performed in this approach and determined the F1 Scores for the Cross Validation Approach. Let us suppose we have n_j examples in S of class y_j such that

$$\sum_j n_j = n$$

. Also lets say the hypersphere has k_j examples of class y_j. IF we want to classify a new example x with its k-nearest neighbors, and keeping the Bayes rule in mind we get:

$$h(x) = arg\max_j \frac{\hat{p}(x|y_j).\hat{p}(y_j)}{\hat{p}(x)} = arg\max_j \frac{\frac{k_j}{V_n \times n_j} \times \frac{n_j}{n}}{\frac{k}{n \times V_n}}$$

$$= arg\max_j \frac{k_j}{k}$$

2. Abstract

We have bifurcated the analysis into four parts, as prompted by the problem statement,

Algorithm 1 Pseudo Code for KNN

Input: **x**, S, d
Output: class of **x**
for $(x,y) \in S$ **do**
  Compute the distance $d(\bar{x}, x)$
end for
Sort the n distances by increasing order;
Count the number of occurrences in each class y_j among the k-nearest neighbors;
Assign to **x** the most frequent class;

- **Tune the best k of a kNN classifier** by cross-validation.
- **Reducing the complexity** by running Data Reduction Algorithms.
- **Speeding up the calculation** by comparing the two methods in terms of time (1NN with brute force).
- **Generating Artificial Imbalance** in the training data and analyzing the impact on the accuracy. **

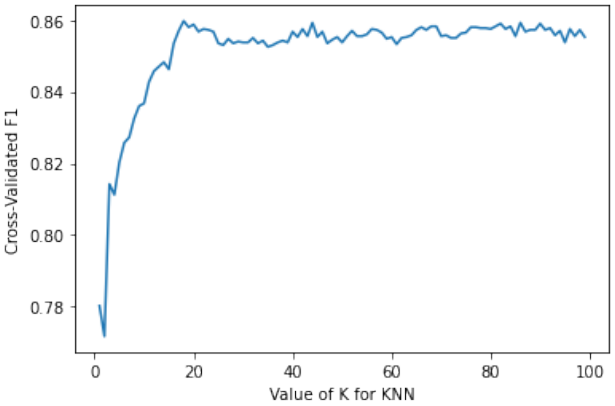

Figure 1. Accuracy vs K before running data reduction

2.1. Hyper-parameter Tuning for k in kNN

We test 100 values of k ranging from 1 to 100 by using 10-fold cross-validation with each k and calculate the cross-validation score for each respective value, this way we are able to determine the best possible value of k along with the accuracy of the classifier. We examine the best F1 Score for the model and then compare it with the hyper-parameter value to maximize the gradient and find an efficient value.

2.2. Reducing the Complexity

To reduce the complexity of the algorithm, we reduce the dimensionality of the data by dropping the misclassified samples and checking the number of good samples to look out for. The whole process is carried out in two steps, first one being data reduction by dropping the misclassified samples, where after splitting our data, we make a prediction with every iteration and identify the misclassified sample and dropping it from the dataset, followed by checking the number of good samples by making a new set of features and values in the new dataset and fitting it into the model to find out the accuracy. You can observe (See figure 2) there is a slight improvement in the overall behavior of noise affecting the dataset resulting in a much smoother curve than before.

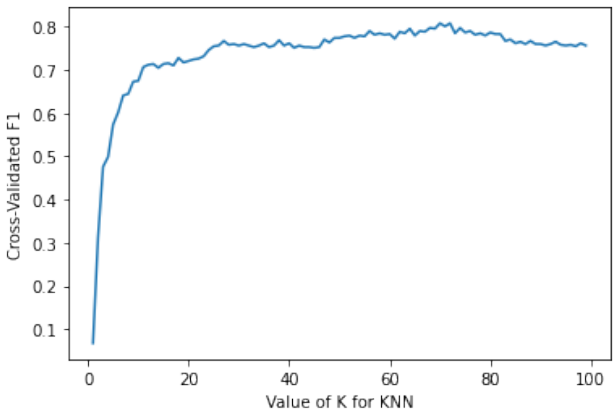

Figure 2. Accuracy vs K after data reduction

2.3. Speeding up the Calculation

We have learned three algorithms to achieve our goal of speeding up the algorithm. i.e. KDTree, BallTree and Brute Force.

To find out which method is the fastest, we have millis functions both before and after the training and testing commands, which are then compared at the end in seconds. A trend is observed when we plot all the results (See Figure 3) That both during training and testing time. KDTree proves to be the fastest and Brute Force, the slowest (comparatively) even though the results were highly influential given that the data was balanced.

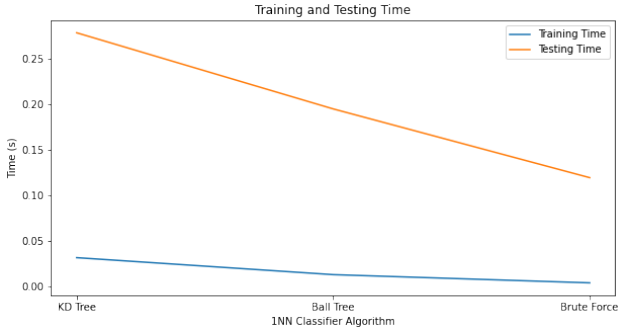

Figure 3. Comparison of all Three methods for 1NN classifier against Time

2.4. Generating Artificial Imbalance

The last and most crucial experiment to be performed on the test data is generating an artificial imbalance, which we can obtain by simply sampling the data throughout the index of our original dataset and using it to calculate different k in models, then return the average accuracy and f1 score. We can see that as the value of k increases, our f1 score decreases, making a downward slope (See figure 4). This in turn is not good for the model.

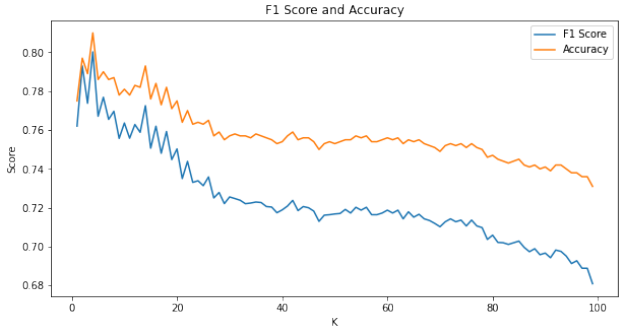

Figure 4. F1 Score and Accuracy after generating an imbalance

3. Conclusion

In conclusion, Imbalanced data can cause models to be biased and show unflattering results, as we can see, but any algorithm can be improved by using the techniques mentioned in this project, to obtain a better result. The results above show us that artificially imbalancing the data lead us to have a negative impact on the model accuracy, and the F-Measure proved to be useful as compared to using

just accuracy. If we look at it from an objective set of eyes, if the Bayes Classification provides a better accuracy, we must use all the resources available to compare it to our own methods.